12-19-2008

Graph Distances in the Data-Stream Model

Joan Feigenbaum
Yale University

Sampth Kannan
University of Pennsylvania, kannan@cis.upenn.edu

Andrew Mcgregor
University of Massachusetts - Amherst

Siddarth Suri
Yahoo Research

Jian Zhang
Louisiana State University

Recommended Citation

Joan Feigenbaum, Sampth Kannan, Andrew Mcgregor, Siddarth Suri, and Jian Zhang, "Graph Distances in the Data-Stream Model", *SIAM Journal of Computing* 38(5), 1709-1727. December 2008. http://dx.doi.org/10.1137/070683155

Graph Distances in the Data-Stream Model

Abstract
We explore problems related to computing graph distances in the data-stream model. The goal is to design algorithms that can process the edges of a graph in an arbitrary order given only a limited amount of working memory. We are motivated by both the practical challenge of processing massive graphs such as the web graph and the desire for a better theoretical understanding of the data-stream model. In particular, we are interested in the trade-offs between model parameters such as per-data-item processing time, total space, and the number of passes that may be taken over the stream. These trade-offs are more apparent when considering graph problems than they were in previous streaming work that solved problems of a statistical nature. Our results include the following: (1) Spanner construction: There exists a single-pass, (O) over tilde (tn(1+1/t))-space, (O) over tilde (t(2)n(1/t))-time-per-edge algorithm that constructs a (2t + 1)-spanner. For t = Omega(log n/log log n), the algorithm satisfies the semistreaming space restriction of O(n polylog n) and has per-edge processing time O(polylog n). This resolves an open question from [J. Feigenbaum et al., Theoret. Comput. Sci., 348 (2005), pp. 207-216]. (2) Breadth-first-search (BFS) trees: For any even constant k, we show that any algorithm that computes the first k layers of a BFS tree from a prescribed node with probability at least 2/3 requires either greater than k/2 passes or Omega(n(1+1/k)) space. Since constructing BFS trees is an important subroutine in many traditional graph algorithms, this demonstrates the need for new algorithmic techniques when processing graphs in the data-stream model. (3) Graph-distance lower bounds: Any t-approximation of the distance between two nodes requires Omega(n(1+1/t)) space. We also prove lower bounds for determining the length of the shortest cycle and other graph properties. (4) Techniques for decreasing per-edge processing: We discuss two general techniques for speeding up the per-edge computation time of streaming algorithms while increasing the space by only a small factor.

Keywords
stream algorithms, graph distances, spanners, COMMUNICATION COMPLEXITY, ALGORITHMS, CONSTRUCTION

SIAM J. COMPUT.
Vol. 38, No. 5, pp. 1709–1727

GRAPH DISTANCES IN THE DATA-STREAM MODEL*

JOAN FEIGENBAUM†, SAMPATH KANNAN‡, ANDREW MCGREGOR§, SIDDHARTH SURI¶, AND JIAN ZHANG‖

Abstract. We explore problems related to computing graph distances in the data-stream model. The goal is to design algorithms that can process the edges of a graph in an arbitrary order given only a limited amount of working memory. We are motivated by both the practical challenge of processing massive graphs such as the web graph and the desire for a better theoretical understanding of the data-stream model. In particular, we are interested in the trade-offs between model parameters such as per-data-item processing time, total space, and the number of passes that may be taken over the stream. These trade-offs are more apparent when considering graph problems than they were in previous streaming work that solved problems of a statistical nature. Our results include the following: (1) *Spanner construction:* There exists a single-pass, $\tilde{O}(tn^{1+1/t})$-space, $\tilde{O}(t^2n^{1/t})$-time-per-edge algorithm that constructs a $(2t+1)$-spanner. For $t = \Omega(\log n/\log\log n)$, the algorithm satisfies the semistreaming space restriction of $O(n \operatorname{polylog} n)$ and has per-edge processing time $O(\operatorname{polylog} n)$. This resolves an open question from [J. Feigenbaum et al., *Theoret. Comput. Sci.*, 348 (2005), pp. 207–216]. (2) *Breadth-first-search (BFS) trees:* For any even constant k, we show that any algorithm that computes the first k layers of a BFS tree from a prescribed node with probability at least 2/3 requires either greater than $k/2$ passes or $\tilde{\Omega}(n^{1+1/k})$ space. Since constructing BFS trees is an important subroutine in many traditional graph algorithms, this demonstrates the need for new algorithmic techniques when processing graphs in the data-stream model. (3) *Graph-distance lower bounds:* Any t-approximation of the distance between two nodes requires $\Omega(n^{1+1/t})$ space. We also prove lower bounds for determining the length of the shortest cycle and other graph properties. (4) *Techniques for decreasing per-edge processing:* We discuss two general techniques for speeding up the per-edge computation time of streaming algorithms while increasing the space by only a small factor.

Key words. stream algorithms, graph distances, spanners

AMS subject classifications. 68Q05, 68Q17, 68Q25, 68W20, 68W25

DOI. 10.1137/070683155

1. Introduction.

In recent years, streaming has become an active area of research and an important paradigm for processing massive data sets [4, 23, 27]. Much of the existing work has focused on computing statistics of a stream of data elements,

*Received by the editors February 20, 2007; accepted for publication (in revised form) August 4, 2008; published electronically December 19, 2008. This work was supported by the DoD University Research Initiative (URI) administered by the Office of Naval Research under grant N00014-01-1-0795. It first appeared in the Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, 2005 [21].

http://www.siam.org/journals/sicomp/38-5/68315.html

†Department of Computer Science, Yale University, P.O. Box 208285, New Haven, CT 06520-8285 (feigenbaum-joan@cs.yale.edu). This author's research was supported in part by ONR grant N00014-01-1-0795 and NSF grant ITR-0331548.

‡Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104 (kannan@cis.upenn.edu). This author's research was supported in part by ARO grant DAAD 19-01-1-0473 and NSF grant CCR-0105337.

§Department of Computer Science, University of Massachusetts, 140 Governors Drive, Amherst, MA 01003-9264. This work was done while the author was at the University of Pennsylvania and was supported in part by NSF grant ITR-0205456.

¶Yahoo! Research, 111 West 40th Street, 17th floor, New York, NY 10018 (suri@yahoo-inc.com). This work was done while the author was at the University of Pennsylvania and was supported in part by NIH grant T32HG000046-05.

‖Department of Computer Science, Louisiana State University, Baton Rouge, LA 70803 (jz@lsu.edu). This work was done while the author was at Yale University and was supported by NSF grant ITR-0331548.

e.g., frequency moments [4, 29], ℓ_p distances [23, 28], histograms [24, 26], and quantiles [25]. More recently, there have been extensions of the streaming research to the study of graph problems [6, 22, 27]. Solving graph problems in this model has raised new challenges, because many existing approaches to the design of graph algorithms are rendered useless by the sequential-access limitation and the space limitation of the streaming model.

Massive graphs arise naturally in many real-world scenarios. Two examples are the *call graph* and the *web graph*. In the call graph, nodes represent telephone numbers and edges correspond to calls placed during some time interval. In the web graph, nodes represent web pages, and the edges correspond to hyperlinks between pages. Also, massive graphs appear in structured data mining, where the relationships among the data items in the data set are represented as graphs. When processing these graphs it is often appropriate to use the streaming model. For example, the graph may be revealed by a web crawler or the graph may be stored on external-memory devices, where being able to process the edges in an arbitrary order improves I/O efficiency. Indeed, the authors of [33] argue that one of the major drawbacks of standard graph algorithms, when applied to massive graphs such as the web, is their need to have random access to the edge set.

In general it seems that most graph algorithms need to access the data in a very adaptive fashion. Since we cannot store the entire graph, emulating a traditional algorithm may necessitate an excessive number of passes over the data. There has been some success in estimating quantities that are of a statistical nature, e.g., counting triangles [6, 11, 31] or estimating frequency and entropy moments of the degrees in a multigraph [12, 15]. However, it seemed for a while that more "complicated" computation was not possible in this model. For example, Buchsbaum, Giancarlo, and Westbrook [10] demonstrated the intrinsic difficulty of computing common neighborhoods in the streaming model with small space. One possible way to ameliorate the situation is to consider algorithms that use $\Theta(n \operatorname{polylog} n)$ space, i.e., space roughly proportional to the number of nodes rather than the number of edges. This space restriction was identified as an apparent "sweet-spot" for graph streaming in a survey article by Muthukrishnan [37] and dubbed the *semistreaming* space restriction. This spurred further research on algorithms for graph problems in the streaming model such as distance estimation [19, 22], matchings [22, 36], and connectivity [21, 41], including the work described here. We will provide further discussion of the results on distance estimation in the next section.

A related model is the *semiexternal model*. This was introduced by Abello, Buchsbaum, and Westbrook [1] for computations on massive graphs. In this model, the vertex set can be stored in memory, but the edge set cannot. However, unlike in our model, random access to the edges, although expensive, was allowed. Finally, graph problems have been considered in a model that extends the stream model by allowing the algorithm to write to the stream during each pass [2, 16]. These *annotations* can then be utilized by the algorithm during successive passes. Aggarwal et al. [2] go further and suggest a model in which *sorting passes* are permitted, and the data-stream is sorted according to a key encoded by the annotations.

Designing algorithms for computing graph distances is a very well studied problem, and graph distances are a natural quantity to study when trying to understand properties of massive graphs such as the diameter of the world wide web [3]. We start with a formal definition of the relevant terms.

DEFINITION 1.1 (graph distance, diameter, and girth). *For an undirected, un-*

weighted graph $G = (V, E)$, we define a distance function $d_G : V \times V \to \{0, \ldots, n-1\}$, where $d_G(u, v)$ is the length of the shortest path in G between u and v. The diameter of G is the length of the longest shortest path, i.e.,

$$\mathrm{Diam}(G) = \max_{u,v \in V} d_G(u, v).$$

The girth of G is the length of the shortest cycle in G, i.e.,

$$\mathrm{Girth}(G) = 1 + \min_{(u,v) \in E} d_{G \setminus (u,v)}(u, v).$$

Classic algorithms such as Dijkstra, Bellman–Ford, and Floyd–Warshall are taught widely [14]. Recent research has focused on computing approximate graph distances [5, 7, 17, 40]. Unfortunately, these algorithms seem to be inherently unsuitable for computing distances in the streaming model; an important subroutine of many of the existing algorithms is the construction of breadth-first-search (BFS) trees, and one of our main results is a lower bound on the computational resources required to compute a BFS tree. For example, Thorup and Zwick provide a construction of distance oracles for approximating distances in graphs [40]. Although *all-pairs*-shortest-path distances can be approximated using this oracle, their oracle construction requires the computation of shortest-path trees for certain vertices. Indeed, many constructions of distance oracles have this requirement, i.e., they need to compute *some* distances between certain pairs of vertices in order to build a data structure from which the all-pairs-shortest-path distances can be approximated.

A common method for approximating graph distances is via the construction of *spanners*.

DEFINITION 1.2 (spanners). *A subgraph $H = (V, E')$ is an (α, β)-*spanner *of $G = (V, E)$ if, for any vertices $x, y \in V$,*

$$d_G(x, y) \le d_H(x, y) \le \alpha d_G(x, y) + \beta.$$

When $\beta = 0$, we call the spanner a multiplicative-spanner *and refer to α as the* stretch factor *of the spanner.*

In [22], a simple semistreaming spanner construction is presented. That algorithm constructs a $(\log n)$-spanner in one pass using $O(n \operatorname{polylog} n)$ space. However, this algorithm needs $O(n)$ time to process each edge in the input stream. Such a per-edge processing time is prohibitive, especially in the streaming model when edges may be arriving in quick succession. The work of [19] studies the construction of $(1 + \epsilon, \beta)$ spanners in the streaming model. However, the algorithm of [19] requires multiple passes over the input stream, while our construction needs only one pass.

Notation and terminology. We refer to an event's occurring "with high probability" if the probability of the event is at least $1 - 1/n^{\Omega(1)}$. We denote the set $\{1, \ldots, t\}$ by $[t]$. Let $\mathcal{P}(S)$ denote the power set of S, i.e., $\{S' : S' \subset S\}$.

1.1. Our results. Our results include the following.

1. *Spanner construction:* There exists a single-pass $O(tn^{1+1/t} \log^2 n)$ space, $O(t^2 n^{1/t} \log n)$ time-per-edge algorithm that constructs a $(2t + 1)$-spanner. For $t = \Omega(\log n / \log \log n)$, the algorithm satisfies the semistreaming space restriction of $O(n \operatorname{polylog} n)$ and has per-edge processing time $O(\operatorname{polylog} n)$. The algorithm is presented in section 3. This result resolves an open question from [22].

2. *BFS trees:* For any even constant k, any algorithm that computes the first k layers of a BFS tree from a prescribed node with probability at least 2/3 requires either greater than $k/2$ passes or $\tilde{\Omega}(n^{1+1/k})$ space. This result is proved in section 4. Since constructing BFS trees is an important subroutine in many traditional graph algorithms, this demonstrates the need for new algorithmic techniques when processing graphs in the data-stream model.
3. *Lower bounds:* In section 5, we present lower bounds for the following problems:
   (a) *Connectivity and other balanced properties:* We show that testing any of a large class of graph properties, which we refer to as *balanced properties*, in one pass requires $\Omega(n)$ space. This class includes properties such as connectivity and bipartiteness. This result provides a formal motivation for the *semistreaming* space restriction, where algorithms are permitted $O(n \operatorname{polylog} n)$ space.
   (b) *Graph distances and graph diameter:* We show that any single-pass algorithm that returns a t-approximation of the graph distance between two given nodes with probability at least 3/4 requires $\Omega(n^{1+1/t})$ bits of space. Furthermore, this bound also applies to estimating the diameter of the graph. Therefore, approximating a distance using the above spanner construction is only about a factor of 2 from optimal in terms of the approximation factor achievable for a given space restriction.
   (c) *Girth:* Any p-pass algorithm that ascertains whether the length of the shortest cycle is longer than g requires $\Omega\left(p^{-1}(n/g)^{1+4/(3g-4)}\right)$ bits of space.
4. *Techniques for decreasing per-edge processing:* In section 6, we present a method for local amortization of per-data-item complexity. We also present a technique for adapting existing partially dynamic graph algorithms to the semistreaming model.

The above results indicate various trade-offs between model parameters and accuracy. These include the smooth trade-off between the space a single-pass algorithm is permitted and the accuracy achievable when estimating graph distances. For multiple-pass algorithms, a smooth trade-off between passes and space is evident when trying to compute the girth of a graph. This trade-off is, in a sense, fundamental as it indicates that the only way to get away with using half the amount of space is essentially to make half as much progress in each pass. The trade-off between space and passes when computing BFS trees indicates that, as we restrict the space, no algorithm can do much better than emulating a trivial traditional graph algorithm and will consequently require an excessive number of passes.

Recent developments. Since the preliminary version of this paper appeared, improved spanner construction algorithms were presented by Baswana [8] and Elkin [18]. Extensions of ideas in section 6 have been developed by Zelke [41, 42].

2. Preliminaries. In this section, we give a formal definition of a graph stream.

DEFINITION 2.1 (graph stream). *For a data-stream $A = \langle a_1, a_2, \ldots, a_m \rangle$, where $a_j \in [n] \times [n]$, we define a graph G on n vertices $V = \{v_1, \ldots, v_n\}$ with edges $E = \{(v_i, v_k) : a_j = (i, k)$ for some $j \in [m]\}$.*

We usually assume that each a_j is distinct, but this assumption is often not necessary. When the data items are not distinct, the model can naturally be extended to consider multigraphs; i.e., an edge (v_i, v_k) has multiplicity equal to $|\{j : a_j = (i, k)\}|$. Similarly, we consider undirected graphs, but the definition can be generalized

to define digraph streams. Sometimes we will consider weighted graphs, and in this case $a_j \in [n] \times [n] \times \mathbb{N}$, where the third component of the data item indicates a weight associated with the edge. Note that some authors have also considered a special case of the model, the adjacency-list model, in which all incident edges are grouped together in the stream [6]. We will be interested in the fully general model.

3. Spanners. In this section, we present a single-pass streaming algorithm that constructs a $(2t+1)$-spanner for an unweighted, undirected graph. The algorithm uses some of the ideas from [7] and adapts them for use in the data-stream model. We then extend this algorithm to construct $(1+\epsilon)(2t+1)$-spanners for weighted, undirected graphs using a geometric grouping technique.

Overview of the algorithm. Intuitively, we want to cover each "dense subgraph" of the graph by a tree of small depth, i.e., one of depth $O(\log n)$, rooted at some node. If there are many edges between two such dense subgraphs, only one representative edge needs to be remembered. Edges between vertices that are not part of such dense subgraphs also need to be remembered, but we will argue that there are not too many of them. The construction requires a delicate balance between trying to include as many nodes as possible in a small number of dense subgraphs and ensuring that the depth of the spanning tree covering each dense subgraph is $O(\log n)$.

Our clusters are similar to those used in [5, 7, 17]. However, the constructions of the clusters in [5, 7, 17] all employ an approach similar to BFS; i.e., the clusters are constructed layer by layer. Such a layer-by-layer process is important to ensure that the clusters constructed in [5,7,17] have small diameters. In the streaming model, this would necessitate multiple passes over the input stream. Our labeling scheme employs a different strategy to control the clusters' diameters, thus bypassing the BFS.

In more detail, in the case in which we are constructing $O(\log n)$-spanners, we consider $\log n/2$ levels. Each node is present at the bottom level and is "promoted" to each successively higher level with probability $1/2$. Whenever a node is present at level i, a cluster at level i is dynamically grown around it as we process the edges. The expected number of clusters at level $i+1$ is half of the expected number of clusters at level i. At the top level, the expected number of clusters is $\sqrt{n}$. As the algorithm goes through the input stream of edges, a node in a lower-level cluster may join a higher-level cluster. The constructed spanner consists of three types of edges: (1) a small-diameter spanning tree for each cluster, (2) one edge between each pair of top-level clusters whose vertex sets have at least one edge between them, and (3) a small number of "other" edges that do not fit into either of these two categories and are necessary to preserve the spanner property.

The above ideas are applicable mutatis mutandis when a t-spanner is sought for other values of t. The description below is for arbitrary t.

The algorithm. A label l used in our construction is a positive integer. Given two parameters n and t, the set of labels L used by our algorithm is generated in the following way. Initially, we have labels $[n]$. We denote by L^0 this set of labels and call them the *level*-0 labels. Independently, and with probability $n^{-1/t}$, each label $l \in L^0$ will be put into a set S^0 and marked as *selected*. From each label l in S^0, we generate a new label $l' = l + n$. We denote by L^1 the set of newly generated labels and call them the *level*-1 labels. We then apply the above selection and new-label-generation procedure on L^1 to get the set of level-2 labels L^2. We continue this until the level-$\lfloor t/2 \rfloor$ labels $L^{\lfloor t/2 \rfloor}$ are generated. If a level-$(i+1)$ label l is generated from a level-i label l', we call l the *successor* of l' and denote it by $\mathrm{Succ}(l') = l$. The set of labels we will use in our algorithm is the union of labels of levels $0, 1, 2, \ldots, \lfloor t/2 \rfloor$, i.e.,

$L = \bigcup_0^{\lfloor t/2 \rfloor} L^i$. Note that L can be generated before the algorithm sees the edges in the stream. But, in order to generate the labels in L, except in the case $t = \Theta(\log n)$, the algorithm needs to know n, the number of vertices in the graph, before seeing the edges in the input stream. For $t = \Theta(\log n)$, a simple modification of the above method can be used to generate L without knowing n, because the probability for a label to be selected is constant.

At first glance, it might appear that our labeling scheme resembles the sequences of sets initially constructed by the algorithm of Thorup and Zwick [40]. In our construction, the generation of the clusters of vertices depends on both the labels and the edge set. Thorup and Zwick, however, generate their sequence of sets independently of the edges. But, this is just the first step in their algorithm. Subsequently, their algorithm computes the exact distance between a fixed vertex and each of the sets of vertices in the sequence. Computing the exact distances between a pair of vertices is difficult in the streaming model. Our algorithm takes a very different approach from [40] to avoid this difficulty.

While going through the stream, our algorithm will label each vertex with labels chosen from L. The algorithm may label a vertex v with multiple labels; however, v will be labeled by at most one label from L^i for $i \in [\lfloor t/2 \rfloor]$. Moreover, if v is labeled by a label l, and l is selected, the algorithm will also label v with the label $\mathrm{Succ}(l)$.

Denote by l^i a label of level i, i.e., $l^i \in L^i$. Let $L(v) = \{l^0, l^{k_1}, l^{k_2}, \ldots, l^{k_j}\}$ be the collection of labels that has been assigned to the vertex v, where $0 < k_1 < \cdots < k_j \leq \lfloor t/2 \rfloor$. Let

$$\mathrm{Height}(v) = \max\{j | l^j \in L(v)\}$$

and $\mathrm{Top}(v) = l^k \in L(v)$ s.t. $k = \mathrm{Height}(v)$. Let $C(l)$ be the collection of vertices that are labeled with the label l.

The sets $L(v)$ and $C(l)$ will grow while the algorithm goes through the stream and labels the vertices. For each $C(l)$, our algorithm stores a tree, $\mathrm{Tree}(l)$, on the vertices of $C(l)$ and the tree is rooted on the first vertex that gets labeled by l. We say an edge (u, v) connects $C(l)$ and $C(l')$ if u is labeled with l and v is labeled with l'. For some pairs of labels $l, l' \in L^{\lfloor t/2 \rfloor}$, our algorithm will store edges that connect $C(l)$ and $C(l')$. We denote by H the set of such edges stored by our algorithm. In addition, for each vertex v, we denote by $M(v)$ the other edges incident to v that are stored by our algorithm. Intuitively, the subgraph induced by $\cup_{v \in V} M(v)$ is the sparse part of the graph G. The spanner constructed by the algorithm is the union of the rooted trees for all the labels, $M(v)$ for all the vertices, and the set H. The detailed algorithm is given in Figure 3.1.

Analysis. We start with two preliminary lemmas showing that, with high probability, the spanner construction requires only a small amount of working space. Note that randomness is introduced in the generation of the labels in Algorithm 1; that is the reason that the bounds stated in both lemmas are true with high probability.

LEMMA 3.1. *With high probability, for all $v \in V$, $|M(v)| = O(tn^{1/t} \log n)$.*

Proof. Let $M^{(i)}(v) \subseteq M(v)$ be the set of edges added to $M(v)$ during the period when $\mathrm{Height}(v) = i$. Let $L(M(v)) = \cup_{(u,v) \in M(v)} L(u)$ be the set of labels that have been assigned to the vertices in $M(v)$. An edge (u, v) is added to $M(v)$ only in step 2(b)ii. Note that, in this case, $\lfloor \frac{t}{2} \rfloor \geq \mathrm{Height}(u) \geq \mathrm{Height}(v)$. Hence, the set $L_v(u)$ is not empty. Also, by the condition in step 2(b)ii, an edge (u, v) is added only when none of the labels in $L_v(u)$ appears in $L(M(v))$. Thus, if we add the edge in this step, we will introduce/add the label(s) in $L_v(u)$ to $L(M(v))$. Because $L_v(u)$

Algorithm 1 (efficient one pass spanner construction).
The input to the algorithm is an unweighted, undirected graph $G = (V, E)$, presented as a stream of edges, and two positive integer parameters n and t.

1. Generate the set L of labels as described. $\forall\ v_i \in V$, label vertex v_i with label $i \in L^0$. If i is selected, label v_i with $\mathrm{Succ}(i)$. Continue this until we encounter a label that is not selected. Set $M(v_i) \leftarrow \emptyset$ and $H \leftarrow \emptyset$.
2. Upon seeing an edge (u, v) in the stream, if $L(v) \cap L(u) \neq \emptyset$, drop the edge. Otherwise, consider the following cases:
   (a) If $\mathrm{Height}(v) = \mathrm{Height}(u) = \lfloor t/2 \rfloor$, and there is no edge in H that connects $C(\mathrm{Top}(v))$ and $C(\mathrm{Top}(u))$, set $H \leftarrow H \cup \{(u, v)\}$.
   (b) Otherwise, assume, without loss of generality, that $\lfloor t/2 \rfloor \geq \mathrm{Height}(u) \geq \mathrm{Height}(v)$. Consider the collection of labels
   $$L_v(u) = \{l^{k_1}, l^{k_2}, \ldots, l^{\mathrm{Height}(u)}\} \subseteq L(u),$$
   where $\mathrm{Height}(v) \leq k_1 \leq k_2 \leq \cdots \leq \mathrm{Height}(u)$. Let $l \in L_v(u)$ be the selected label whose level is the lowest among the selected labels in $L_v(u)$.
      i. If such a label l exists, label the vertex v with the successor $l' = \mathrm{Succ}(l)$ of l, i.e.,
      $$L(v) \leftarrow L(v) \cup \{l'\}.$$
      Incorporate the edge in the rooted tree $\mathrm{Tree}(l')$. If l' is selected, label v with $l'' = \mathrm{Succ}(l')$ and incorporate the edge in the tree $\mathrm{Tree}(l'')$. Continue until we see a label that is not marked as selected. (Note that $\mathrm{Height}(v)$ is increased by this process.)
      ii. If no such label l exists and there is no edge (u', v) in $M(v)$ such that u, u' are labeled with the same label $l \in L_v(u)$, add (u, v) to $M(v)$.
3. After seeing all the edges in the stream, output the union of the rooted trees for all the labels, $M(v)$ for all the vertices, and the set H as the spanner.

FIG. 3.1. *An efficient, one-pass algorithm for computing sparse spanners. See the descriptions before Lemma* 3.1 *for the definitions of* H, $M(v)$, $L(v)$, $C(l)$, $\mathrm{Tree}(l)$, $\mathrm{Succ}(l)$, $\mathrm{Top}(v)$, *and* $\mathrm{Height}(v)$.

is nonempty, it adds at least one new label to $L(M(v))$. This is the case for every edge in $M^{(i)}(v)$. Let B be the set of distinct labels that $M^{(i)}(v)$ adds to $L(M(v))$. Furthermore, because each edge in $M^{(i)}(v)$ adds at least one new label, the size of B is at least $|M^{(i)}(v)|$. Note that the labels in B are not marked as selected. Otherwise, the algorithm would have taken step 2(b)i instead of step 2(b)ii. Hence, the size of B satisfies

$$\Pr(|B| = k) \leq (1 - 1/n^{1/t})^k.$$

Thus, with high probability, $|M^{(i)}(v)| = O(n^{1/t} \log n)$. Because i can take only $O(t)$ values, with high probability,

$$|M(v)| = \sum_i |M^{(i)}(v)| = O(tn^{1/t} \log n). \qquad \square$$

LEMMA 3.2. *With high probability, the algorithm stores* $O(tn^{1+1/t}\log n)$ *edges.*

Proof. The algorithm stores edges in the set H, in the rooted trees for each cluster $C(l)$, and in the sets $M(v)$, for all $v \in V$. By the Chernoff bound and the union bound, with high probability the number of clusters at level $t/2$ is $O(\sqrt{n})$, and the size of the set H is $O(n)$.

For each label l, the algorithm stores a rooted tree for the set of vertices $C(l)$. The rooted tree is formed by the set of edges added when the vertices in $C(l)$ get labeled by the label l. This happens in step 2(b)i. In this step, we add an edge only at the time when a vertex becomes a member of $C(l)$. Therefore, the set of edges forms a tree on $C(l)$. We call this tree the rooted tree of $C(l)$. Note that two vertices u and v in $C(l)$ may share some other label l' of a different level. In this case, they both also belong to $C(l')$. The tree of $C(l')$ may have a path connecting u and v. Hence the subgraph of the spanner induced by the vertices in $C(l)$ is not necessarily a tree. When we say "the rooted tree of $C(l)$," we refer only to the edges added when the vertices in $C(l)$ get labeled by l.

Note that, for each level $i \in [\lfloor t/2 \rfloor]$, a vertex is labeled with at most one label in L^i. Hence, $\sum_{l \in L^i} |C(l)| \leq |V|$. Thus, the number of edges summed over the rooted trees for the labels at level i is $O(n)$, and the total number of edges in all the rooted trees is $O(tn)$. Finally, by Lemma 3.1, with high probability, $|M(v)| = O(tn^{1/t}\log n)$. By the union bound, with high probability, $\sum_{v \in V} |M(v)| = O(tn^{1+1/t}\log n)$. ∎

THEOREM 3.3. *Let G be an unweighted graph. There exists a single-pass $O(tn^{1+1/t}\log^2 n)$ space algorithm that constructs a $(2t+1)$-spanner of G with high probability and processes each edge in $O(t^2 n^{1/t}\log n)$ time.*

Proof. Consider Algorithm 1 in Figure 3.1. At the beginning of the algorithm, for all the labels $l \in L^0$, $C(l)$ is a singleton set, and the depth of the rooted tree for $C(l)$ is zero. We now bound, for label l^i, where $i > 0$, the depth of the rooted tree T^i on the vertices in $C(l^i)$. A tree grows when an edge (u, v) is incorporated into the tree in step 2(b)i. In this case, l^i is a successor of some label l^{i-1} of level $i-1$. Assume that the depth $d_{T^i}(v)$ of the vertex v in the tree is one more than the depth $d_{T^i}(u)$ of the vertex u. Then $u \in C(l^{i-1})$, and the depth $d_{T^{i-1}}(u)$ of u in the rooted tree T^{i-1} of $C(l^{i-1})$ is the same as $d_{T^i}(u)$. Hence, $d_{T^i}(v) = d_{T^{i-1}}(u) + 1$, where T^i is a tree of level i, and T^{i-1} is a tree of level $i-1$. Given that $d_{T^0}(x) = 0$ for all $x \in V$, the depth of a rooted tree for $C(l)$, where l is a label of level i, is at most i.

We proceed to show that, for any edge that the algorithm does not store, there is a path of length at most $2t+1$ that connects the two endpoints of the edge. The algorithm ignores three types of edges. First, if $L(u) \cap L(v) \neq \emptyset$, the edge (u, v) is ignored. In this case, let l be one of the label(s) in $L(u) \cap L(v)$. The nodes u and v are both on the rooted tree for $C(l)$; hence, there is a path of length at most t connecting u and v. Second, (u, v) will be ignored if $\text{Height}(v) = \text{Height}(u) = \lfloor t/2 \rfloor$, and there is already an edge connecting $C(\text{Top}(u))$ and $C(\text{Top}(v))$. In this case, the path connecting u and v has length at most $2t+1$. Finally, in step 2(b)ii, (u, v) will be ignored if there is already another edge in $M(v)$ that connects v to some $u' \in C(l)$, where $l \in L(u)$. Note that u and u' are both on the rooted tree of $C(l)$. Hence, there is a path of length at most $t+1$ connecting u and v.

Hence, the stretch factor of the spanner constructed by Algorithm 1 is $2t+1$. By Lemma 3.2, with high probability, the algorithm stores $O(tn^{1+1/t}\log n)$ edges and requires $O(tn^{1+1/t}\log^2 n)$ bits of space. Also note that the bottleneck in the processing of each edge lies in step 2(b)ii, where, for each label in $L_v(u)$, we need to examine the whole set of $M(v)$. This takes $O(t^2 n^{1/t}\log n)$ time. ∎

Extensions. Once the spanner is constructed, all-pairs-shortest-distances of the graph can be computed from the spanner. This computation does not need to access the input stream and thus can be viewed as postprocessing. We also note that the above algorithm can be used to construct a spanner of a weighted graph $G = (V, E)$ using a geometric grouping technique [13,22]. Namely, we can round each edge weight ω' up to $\min\{\omega(1+\epsilon)^i : i \in \mathbb{Z}, \omega(1+\epsilon)^i \geq \omega'\}$, where ω is the weight of the first edge, and $\epsilon > 0$ is a user-defined accuracy parameter. Let $G^i = (V, E^i)$ be the graph formed from G by removing all edges not of weight $\omega(1+\epsilon)^i$. For each G^i, we construct a spanner in parallel and take the union of these spanners. This leads to the following theorem.

THEOREM 3.4. *Let G be a weighted graph and W be the ratio between the maximum and minimum weights. There exists a single-pass $O(\epsilon^{-1} t n^{1+1/t} \log W \log^2 n)$ space algorithm that constructs a $(1+\epsilon)(2t+1)$-spanner of G with high probability and processes each edge in $O(t^2 n^{1/t} \log n)$ time.*

In the case where $t = \log n / \log\log n$, Algorithm 1 computes a $(2\log n/\log\log n + 1)$-spanner in one pass using $O(n \log^4 n)$ bits of space and processing each edge in $O(\log^4 n)$ time. This answers an open question we posed in [22]. Finally, note that constructing a $(2t+1)$-spanner gives a $(2t+1)$-approximation for the diameter and, indirectly, a $(2t+2)/3$-approximation of the girth. The diameter result is immediate. For the girth approximation, note that, if the constructed spanner is a strict subgraph of G, then the girth of G must have been between 3 and $2t+2$.

4. BFS trees lower bound. In this section, we prove a lower bound on the number of passes required to construct the first l layers of a BFS tree in the streaming model. The result is proved using a reduction from the communication-complexity problem "multivalued pointer chasing." This is a natural generalization of the pointer-chasing problem considered by Nisan and Wigderson [38].

Overview of proof. Nisan and Wigderson [38] considered the problem in which Alice and Bob have functions $f_A : [m] \to [m]$ and $f_B : [m] \to [m]$, respectively. The k-round pointer-chasing problem is to output the result of starting from 1 and alternatively applying f_A and f_B a total of k times, starting with f_A. Nisan and Wigderson proved that, if Bob speaks first, the communication complexity of any k-round communication protocol to solve this problem is $\Omega(m/k^2 - k\log m)$. Jain, Radhakrishnan, and Sen [30] gave a direct sum extension showing that, if there are d pairs of functions and the goal is to perform k-round pointer chasing as above on each pair, the communication-complexity lower bound is approximately d times the bound of [38]. More precisely, they showed a lower bound of $\Omega(dm/k^3 - dk\log m - 2d)$ for the problem.

We show how the lower bound of [30] implies a lower bound on the communication complexity of pointer chasing with "d-valued functions," i.e., functions that map $i \in [m]$ to a subset of $[m]$ of size at most d. If f_A and f_B are such functions, then the result of pointer chasing starting from 1 produces a set of size at most d^k. The key difference between this problem and the problem of [30] is that in the latter, one is concerned only with chasing "like" pointers. That is, if one gets to an element j using the function $f_{A,i}$, one can continue only with $f_{B,i}$. (We will present an example after the formal definition of the appropriate terms.) Nevertheless, we give a reduction that shows that the two problems have fairly similar communication complexity.

Finally, we create a an l-layered graph in which alternate layers have edges corresponding to d-valued functions f_A and f_B. In order to construct the BFS tree, we must solve the l-round, d-valued pointer-chasing problem and then apply the afore-

mentioned lower bound. This will lead to the following theorem.

THEOREM 4.1 (BFS lower bound). *For any constant k, any algorithm that computes the first k layers of a BFS tree with probability at least $2/3$ requires either $k/2$ passes or $\Omega(n^{1+1/k}/(\log n)^{1/k})$ space.*

We now present the above argument formally. Let F_d be the set of functions $f : \mathcal{P}([m]) \to \mathcal{P}([m])$ such that

$$\forall i \in [m],\ |f(i)| \le d \quad \text{and} \quad \forall A \subset [m],\ f(A) = \bigcup_{i \in A} f(i).$$

Throughout the rest of this section, we abuse notation and denote the singleton set $\{i\}$ by i when it appears as the input or the output of a function $f \in F_d$.

DEFINITION 4.2. *Define $g^{k,d} : F_d \times F_d \to \mathcal{P}([m])$ inductively as*

$$g^{0,d}(f_A, f_B) = \{1\} \quad \textit{and} \quad g^{i,d}(f_A, f_B) = \begin{cases} f_A(g^{i-1,d}(f_A, f_B)) & \textit{if } i \textit{ odd,} \\ f_B(g^{i-1,d}(f_A, f_B)) & \textit{if } i \textit{ even.} \end{cases}$$

Define $h^{k,d}$ as the d-fold direct sum of $g^{k,1}$, i.e.,

$$h^{k,d}(\langle f_{A,1}, \ldots, f_{A,d}\rangle, \langle f_{B,1}, \ldots, f_{B,d}\rangle) = \langle g^{k,1}(f_{A,1}, f_{B,1}), \ldots, g^{k,1}(f_{A,d}, f_{B,d})\rangle.$$

EXAMPLE 4.3. *Consider $f_{A,1}, f_{A,2}, f_{B,1}, f_{B,2} \in F_1$, where*

$$\begin{array}{llll} f_{A,1}: 1 \to 1 & f_{A,2}: 1 \to 2 & f_{B,1}: 1 \to 1 & f_{B,2}: 1 \to 3 \\ \quad 2 \to 2 & \quad 2 \to 3 & \quad 2 \to 2 & \quad 2 \to 1 \\ \quad 3 \to 3, & \quad 3 \to 1, & \quad 3 \to 3, & \quad 3 \to 2. \end{array}$$

Let $f_A, f_B \in F_2$ be defined by $f_A(j) := f_{A,1}(j) \cup f_{A,2}(j)$ and $f_B(j) := f_{B,1}(j) \cup f_{B,2}(j)$. Then

$$h^{2,2}(\langle f_{A,1}, f_{A,2}\rangle, \langle f_{B,1}, f_{B,2}\rangle) = \langle 1, 1\rangle, \textit{ whereas } g^{2,2}(f_A, f_B) = \{1, 2, 3\}.$$

Let Alice have function f_A and Bob have function f_B. Let $R^r_\delta(g_{d,k})$ be the r-round randomized communication complexity of $g_{d,k}$ where Bob speaks first, i.e., the number of bits sent in the worst case (over all inputs and random coin tosses) by the best r-round protocol Π in which, with probability at least $1-\delta$, both Alice and Bob learn $g_{d,k}$. The following theorem for $h^{k,d}$ was proved in [30] using an information-theoretic argument in combination with a result by Nisan and Wigderson [38].

THEOREM 4.4 (see Jain, Radhakrishnan, and Sen [30]). $R^k_{1/3}(h^{k,d}) = \Omega(dmk^{-3} - dk \log m - 2d)$.

We now use the above result to prove a bound on the communication complexity of $g^{k,d}$.

THEOREM 4.5. $R^k_{1/3}(g^{k,d}) = \Omega(dm/(k+1)^3 - d(k+1)\log m - 2d - 6d^{k+1} \lg m)$ *if k is even.*

Proof. The proof uses a reduction from $h^{k,d}$. Let $(\langle f_{A,1}, \ldots, f_{A,d}\rangle, \langle f_{B,1}, \ldots, f_{B,d}\rangle)$ be an instance of $h^{k,d}$. Define f^*_A and f^*_B as follows:

$$f^*_A(j) := \{f_{A,i}(j) : i \in [d]\} \text{ and } f^*_B(j) := \{f_{B,i}(j) : i \in [d]\}.$$

Assume that there exists a k-round protocol Π for $g^{k,d}$ that fails with probability at most $1/3$ and communicates $o(dm/(k+1)^3 - d(k+1)\log m - 2d - 12d^{k+1} \lg m - km)$ bits in the worst case. We will show how to transform Π into a $(k+1)$-round protocol

Π' for $h^{k+1,d}$ that fails with probability at most $1/3$ and communicates $o(dm/(k+1)^3 - d(k+1)\log m - 2d)$ bits in the worst case. This contradicts Theorem 4.4 and hence shows that there is no such protocol Π.

Let Π' be a protocol that simulates Π and, in addition, on the jth message $(1 < j \le k+1)$, sends the following set of triples:[1]

$$T_{j-1} = \{\langle a, b, f_{C,a}(b)\rangle : b \in g^{j-2,d}(f_A^*, f_B^*)\}, \quad \text{where} \quad C = \begin{cases} A & \text{if } j \text{ is even,} \\ B & \text{if } j \text{ is odd.} \end{cases}$$

Π is shown to be correct by an inductive argument and requires at most $3d^{j-1}\log m$ additional bits of communication per message, because $b \in g^{j-2,d}(f_A^*, f_B^*)$ and $|g^{j-2,d}(f_A^*, f_B^*)| \le d^{j-2}$, $a \in [d]$, and each $\langle a, b, f_{C,a}(b)\rangle$ can be encoded with at most $3\log m$ bits. However, if Π is successful, then the player who sends the kth message (which is Alice by assumption that k is even and Bob speaks first) of Π also knows $g^{k,d}(f_A^*, f_B^*)$. Hence, she can also send

$$T_{k+1} = \{\langle a, b, f_{A,a}(b)\rangle : b \in g^{k,d}(f_A^*, f_B^*)\}.$$

Hence, after $(k+1)$ rounds, $\bigcup_{i\in[k+1]} T_i$ is known to both parties with probability at least $2/3$ and can be used to deduce $g^{k+1,d}$. The total amount of extra communication required to transmit $\bigcup_{i\in[k+1]} T_i$ is $\sum_{i\in[k+1]} 3d^i \log m \le 6d^{k+1}\log m$. Hence Π' communicates $o(dm/(k+1)^3 - d(k+1)\log m - 2d)$ bits in the worst case. ∎

We are now in a position to prove Theorem 4.1.

Proof. The proof is a reduction from d-valued pointer chasing. Let $m = n/(k+1)$, and let $d = c(m/\log m)^{1/k}$ for some small constant c. Then, since k is constant by Theorem 4.5, $R^k_{1/3}(g^{k,d}) = \Omega(n^{1+1/k}/(\log n)^{1/k})$.

Consider an instance (f_A, f_B) of $g^{k,d}$. The graph described by the stream is on the following set V of $n = (k+1)m$ nodes:

$$V = \bigcup_{0\le i\le k} \{v_1^i, \ldots, v_m^i\}.$$

For $i \in [k]$, we define a set of edges $E(i)$ between $\{v_1^{i-1}, \ldots, v_m^{i-1}\}$ and $\{v_1^i, \ldots, v_m^i\}$ in the following way:

$$E(i) = \begin{cases} \{(v_j^{i-1}, v_\ell^i) : \ell \in f_A(j)\} & \text{if } i \text{ is odd,} \\ \{(v_j^{i-1}, v_\ell^i) : \ell \in f_B(j)\} & \text{if } i \text{ is even.} \end{cases}$$

Suppose there exists an algorithm $\mathcal{A}$ that computes the first k layers of the BFS tree from v_1^1 in p passes using memory M. Let L_r be set of nodes that are at distance exactly r from v_1^1. Note that, for all $r \in [k]$,

$$g^{r,d} = L_r \cap \{v_1^r, \ldots, v_m^r\}.$$

By simulating $\mathcal{A}$ on a stream starting with $\bigcup_{0\le i\le k:\text{even}} E(i)$ and concluding with $\bigcup_{i\in[k]:\text{odd}} E(i)$ in the natural way, we deduce that there exists a $(2p)$-round communication protocol for $g^{k,d}$ that uses only $2pM$ communication. (Note that $2p$ rounds of communication rather than $(2p-1)$ rounds are required, because we required both parties to learn $g^{k,d}$.) Hence, either $2p > k$ or $M = \Omega(n^{1+1/k})$. ∎

[1] Note that on the $(k+1)$th message there is no message of Π to simulate and only T_k is transmitted.

5. Other lower bounds. In this section, we present lower bounds on the space required to estimate graph distances, test whether a graph is connected, and compute the girth of a graph. Our lower bounds are reductions from problems in communication complexity. In the SET-DISJOINTNESS problem, Alice has a length-n binary string $x \in \{0,1\}^n$, and Bob has a length-n binary string $y \in \{0,1\}^n$, where $\sum_{i\in[n]} x_i = \sum_{i\in[n]} y_i = \lfloor n/4 \rfloor$. If Bob is to compute

$$\text{SET-DISJOINTNESS}(x,y) = \begin{cases} 1 & \text{if } x \cdot y = 0, \\ 0 & \text{if } x \cdot y \geq 1 \end{cases}$$

(where $x \cdot y = \sum_{i\in[n]} x_i y_i$) with probability at least 3/4, then it is known that $\Omega(n)$ bits must be communicated between Alice and Bob [32, 39]. In the INDEX problem, Alice has $x \in \{0,1\}^n$, and Bob has $j \in [n]$. If Bob is to compute $\text{INDEX}(x,j) = x_j$ with probability at least 3/4 after a single message from Alice, then it is known that this message must contain $\Omega(n)$ bits (e.g., [34]). To relate our graph-stream problems to these communication problems, we use reductions based upon results from random-graph theory and extremal combinatorics.

5.1. Connectivity and balanced properties. Our first result shows that a large class of problems, including connectivity, cannot be solved by single-pass streaming algorithms in small space. Specifically, we identify a general type of graph property[2] and show that testing any such graph property requires $\Omega(n)$ space.

DEFINITION 5.1 (balanced properties). *We say a graph property $\mathcal{P}$ is* balanced *if there exists a constant $c > 0$ such that, for all sufficiently large n, there exists a graph $G = (V,E)$ with $|V| = n$ and $u \in V$ such that*

$$\min\{|\{v : (V, E \cup \{(u,v)\}) \text{ has } \mathcal{P}\}|, |\{v : (V, E \cup \{(u,v)\}) \text{ has } \neg\mathcal{P}\}| \} \geq cn.$$

In other words, there are $\Omega(n)$ vertices v such that $(V, E \cup \{(u,v)\})$ has $\mathcal{P}$ and $\Omega(n)$ vertices v such that $(V, E \cup \{(u,v)\})$ does not have $\mathcal{P}$.

Many interesting properties are balanced, including connectivity, bipartiteness, and whether there exists a vertex of a certain degree.

THEOREM 5.2. *Testing for any balanced graph property $\mathcal{P}$ with probability 3/4 in a single pass requires $\Omega(n)$ space.*

Proof. Let c be a constant, $G = (V,E)$ be a graph on n vertices, and $u \in V$ be a vertex satisfying the relevant conditions. The proof is by a reduction to the communication complexity of INDEX. Let $(x,j) \in \{0,1\}^{cn} \times [cn]$ be an instance of INDEX. Let $G(x)$ be a relabeling of the vertices of G such that $u = v_n$ and, for $i \in [cn]$, $(V, E \cup \{(v_n, v_i)\})$ has $\mathcal{P}$ if and only if $x_i = 1$. Such a relabeling is possible, because $\mathcal{P}$ does not depend on the labeling of the vertices. Let $e(j) = (v_j, v_n)$. Hence the graph determined by the edges of $G(x)$ and $e(j)$ has $\mathcal{P}$ if and only if $x_j = 1$. Therefore, any single-pass algorithm for testing $\mathcal{P}$ using M bits of work space gives rise to a one-message protocol for solving INDEX, and this implies that $M = \Omega(n)$. ∎

For some balanced graph properties, the above theorem can be generalized. For example, it is possible to show that any p-pass algorithm that determines whether a graph is connected requires $\Omega(np^{-1})$ bits of space [27].

[2] A *graph property* is a boolean function whose inputs are the elements of the adjacency matrix of the graph but whose output is independent of the labeling of the nodes of the graph.

5.2. Graph distances and graph diameter. If we are interested only in estimating the distance between two nodes u and v, it may appear that constructing a graph spanner that gives no special attention to u or v, but rather approximates *all* distances, is an unnecessarily crude approach. In this section, however, we show that the spanner-construction approach yields an approximation at most a factor 2 from optimal. Our result is a generalization of one in [22] that applied to the semistreaming case. Integral to our proof is the notion of a *k-critical edge*.

DEFINITION 5.3. *In a graph $G = (V, E)$, an edge $e = (u, v) \in E$ is* k-critical *if* $d_{G\setminus(u,v)}(u, v) \geq k$.

In Lemma 5.4, we show the existence of a graph G with a large subset of edges E' such that each edge in E' is k-critical, but the removal of all edges in E' leaves a graph with relatively small diameter. The proof uses a probabilistic method.

LEMMA 5.4. *For sufficiently large n and $3 \leq k = o(\log n/\log\log n)$, there exists a set E of edges partitioned into two disjoint sets E_1 and E_2 on a set of n nodes V such that*

1. $|E_2| = \lceil n^{1+1/k}/144 \rceil$,
2. *every edge in E_2 is k-critical for $G = (V, E)$, and*
3. $\mathrm{Diam}(G_1) \leq k + 1$, *where $G_1 = (V, E_1)$.*

Proof. Let $\gamma = 1/k$. Consider choosing a random graph $G' = (V, E')$ on n nodes where each edge is (independently) present with probability $p = 1/(2n^{1-\gamma})$. This is commonly denoted as $G' \sim \mathcal{G}_{n,p}$. We will then construct $G_1 = (V, E_1)$ by deleting each edge in G' with probability $1/2$. We will show that, with nonzero probability, the sets E_1 and $E_2 = \{e \in E' \setminus E_1 : e \text{ is } k\text{-critical for } G'\}$ satisfy the three required properties.

The second property is satisfied by construction. It follows from the fact that, if an edge is k-critical in a graph G, then it is also k-critical in any subgraph of G. We now argue that the third property is satisfied with probability at least $99/100$. First note that the process that generates G_1 is identical to picking $G_1 \sim \mathcal{G}_{n,p/2}$. It can be shown that, with high probability, the diameter of such a graph is less than $1/\gamma + 1$ for sufficiently large n [9, Corollary 10.12].

We now show that the first property is satisfied with probability at least $8/100$. Applying the Chernoff bound and the union bound proves that, with probability at least $99/100$, the degree of every vertex in G' is between $n^\gamma/4$ and $3n^\gamma/4$.

Now consider choosing a random graph and a random edge in that graph simultaneously, i.e., $G' = (V, E') \sim \mathcal{G}_{n,p}$ and an edge $(u, v) \in_R E'$. We prove a lower bound on the probability that (u, v) is k-critical in G'. Let $\Gamma_i(v) = \{w \in V : d_{G'\setminus(u,v)}(v, w) \leq i\}$. For sufficiently large n, conditioned on the event that the maximum degree is at most $3n^\gamma/4$,

$$|\Gamma_k(v)| \leq \sum_{0\leq i\leq k} (3n/4)^{i\gamma} \leq 1.01(3n/4)^{k\gamma} \leq 4(n-1)/5.$$

As G' varies over all possible graphs, by symmetry, each vertex is equally likely to be in $\Gamma_k(v)$. Thus the probability that u is not in this set is at least $1/5$. By Markov's inequality,

$$\Pr\left(|\{(u, v) \in E' : d_{G'\setminus(u,v)}(u, v) \geq k\}| \geq |E'|/9\right) \geq 1/10.$$

Note that, if the degree of every vertex in G' is at least $n^\gamma/4$, then $|E'| \geq n^{1+\gamma}/8$. Hence,

$$\Pr\left(|\{(u, v) \in E' : d_{G'\setminus(u,v)}(u, v) \geq k\}| \geq n^{1+\gamma}/72\right) \geq 9/100.$$

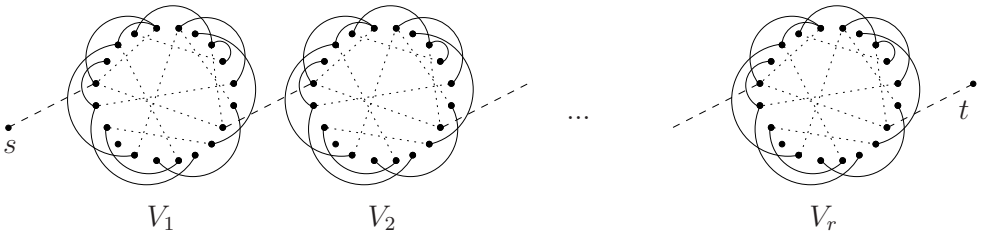

FIG. 5.1. *Diameter lower-bound construction. Edges E_x are dotted, E_j are dashed, and E_m are solid.*

Given that each edge in E' is deleted independently with probability $1/2$ to form E_1, by a further application of the Chernoff bound we deduce that

$$\Pr\left(|\{(u,v) \in E' \setminus E_1 : d_{G' \setminus (u,v)}(u,v) \geq k\}| \geq n^{1+\gamma}/144\right) \geq 8/100.$$

From this set of k-critical edges, we can choose a subset whose size is exactly $\lceil n^{1+\gamma}/144 \rceil$, as required by statement 1. Therefore, all three properties hold with probability at least $1 - 92/100 - 1/100 = 7/100$. □

THEOREM 5.5. *For $3 \leq k = o(\log n / \log\log n)$, any single-pass algorithm that, with probability at least $3/4$, returns $\tilde{D}$ such that*

$$\mathrm{Diam}(G) \leq \tilde{D} \leq (k-1)\,\mathrm{Diam}(G),$$

where G is a weighted graph on n nodes, requires $\tilde{\Omega}(n^{1+1/k})$ space.

Proof. Let $(x, j) \in \{0,1\}^t \times [t]$ be an instance of the INDEX problem. We will show how to transform an algorithm $\mathcal{A}$ for approximating the diameter of a graph into a protocol for INDEX.

Let $G = (V, E = E_1 \cup E_2)$ be a graph on $n' = (144t)^{1/(1+\gamma)}$ nodes with the properties listed in Lemma 5.4. We assume that both Alice and Bob know G and that, moreover, they agree on an ordered list $e_1, \ldots, e_t$ of the edges that are in E_2. This may be assumed, because Alice and Bob can generate identical enumerations of all graphs on n' nodes and all partitions of the edges of each graph into E_1 and E_2, test each graph and partition for the necessary properties, and use the first that passes all of the tests. Finding such a G may take exponential time, but that is all right, because it is only the communication complexity of the resulting INDEX protocol, not the time complexity, that concerns us.

Alice forms the graph $G_x = (V, E_m \cup E_x)$, where $E_x = \{e_i \in E_2 : x_i = 1\}$ and $E_m = E_1$. She then creates the prefix of a stream by taking r (to be determined later) copies of G_x, i.e., a graph on $n'r$ vertices $\{v_1^1, \ldots, v_{n'}^1, v_1^2, \ldots, v_{n'}^2, v_1^3, \ldots, v_{n'}^r\}$ and with edge set $\{(v_j^i, v_k^i) : i \in [r], (v_j, v_k) \in E_x\}$. All these edges have unit weight.

Let j be the index in the instance of INDEX, and let $e_j = (a, b)$. Bob determines the remaining edges E_j as follows: $r-1$ edges of zero weight, $\{(v_b^i, v_a^{i+1}) : i \in [r-1]\}$, and two edges of weight $k+1$, (s, v_a^1) and (v_b^r, t). See Figure 5.1 for a diagram of the construction.

Note that, regardless of the values of x and j, the diameter of the graph described by the stream equals $d_G(s,t)$. Note that $x_j = 1$ implies that $d_G(s,t) = r + 2k + 2$. However, if $x_j = 0$, then $d_G(s,t) = kr + 2k + 2$. Hence, for $r = 2k^2$, the ratio between $kr + 2k + 2$ and $r + 2k + 2$ is at least $k - 1$. Therefore, any single-pass algorithm that approximates the diameter to within a factor of $k - 1$ gives rise to a one-way protocol for solving INDEX. This implies that any such algorithm requires

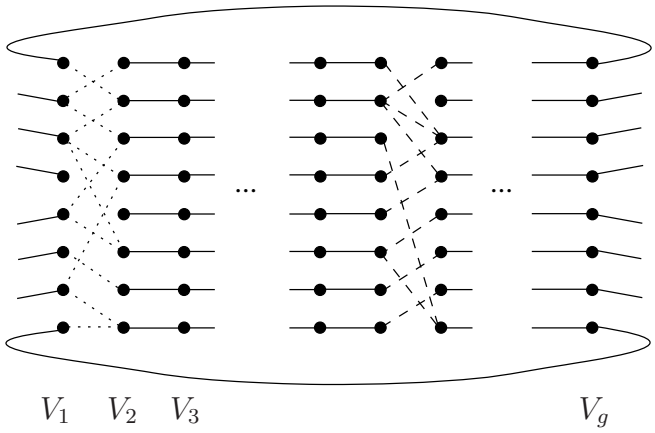

FIG. 5.2. *Girth lower-bound construction. Edges E_x are dotted, E_y are dashed, and E_m are solid.*

$\Omega(n^{1+1/k})$ bits of space, because the total number of nodes in the construction is $n = O((144t)^{1/(1+1/k)}k^2)$. □

5.3. Girth. In this section, we prove a lower bound on the space required by a (multipass) algorithm that tests whether a graph has girth at most g. We shall make use of the following result from [35].

LEMMA 5.6 (see Lazebnik, Ustimenko, and Woldar [35]). *Let $k \geq 1$ be an odd integer, $t = \lfloor \frac{k+2}{4} \rfloor$, and q be a prime power. There exists a bipartite, q-regular graph with at most $2q^{k-t+1}$ nodes and girth at least $k+5$.*

The following lower bound is established with a construction based on Lemma 5.6 that yields a reduction from SET-DISJOINTNESS to girth estimation.

THEOREM 5.7. *For $g \geq 5$, any p-pass algorithm that tests whether the girth of an unweighted graph is at most g requires $\Omega\left(p^{-1}(n/g)^{1+4/(3g-4)}\right)$ space. If g is odd, this can be strengthened to $\Omega\left(p^{-1}(n/g)^{1+4/(3g-7)}\right)$ space.*

Proof. Let q be a prime power; let $k = g-4$ if g is odd, and $k = g-3$ if g is even. Let $t = \lfloor \frac{k+2}{4} \rfloor$. Then,

$$k - t + 1 \leq k - \frac{k+2}{4} + 3/4 + 1 \leq \begin{cases} (3g-7)/4 & \text{if } g \text{ is odd,} \\ (3g-4)/4 & \text{if } g \text{ is even.} \end{cases}$$

Lemma 5.6 implies that there exists a q-regular graph $G' = (L \cup R, E')$ with at most $2n' \leq 2q^{k-t+1}$ nodes and girth at least $g+1$. Let $L = \{l_1, \ldots, l_{n'}\}$ and $R = \{r_1, \ldots, r_{n'}\}$ and, for each $i \in [n']$, $D_i = \Gamma(l_i)$.

We let $(x, y) \in \{0,1\}^r \times \{0,1\}^r$ be an instance of SET-DISJOINTNESS where $r = n'q$. It will be convenient to write $x = x^1 \ldots x^{n'}$ and $y = y^1 \ldots y^{n'}$, where $x^i, y^j \in \{0,1\}^q$. We will show how to transform a p-pass algorithm $\mathcal{A}$ for testing whether the girth of a graph is at most g into a protocol for SET-DISJOINTNESS. If $\mathcal{A}$ uses M bits of working memory, then the protocol will transmit $O(pM)$ bits. Hence $M = \Omega(p^{-1}n'q)$.

Alice and Bob construct a graph G based upon G', x, and y as follows. For $i \in [g]$, let $V_i = \{v_1^i, \ldots, v_{n'}^i\}$. For each $i \in [n']$, let $D_i(x) \subset D_i$ be the subset of D_i whose characteristic vector is x^i. $D_i(y)$ is defined similarly. There are three sets of edges on

these nodes:

$$E_m = \bigcup_{j=\lfloor g/2 \rfloor + 1}^{g} \{(v_i^j, v_i^{j+1}) : i \in [n']\},$$
$$E_x = \{(v_i^1, v_j^2) : j \in D_i(x), i \in [n']\}, \text{ and}$$
$$E_y = \{(v_j^{\lfloor g/2 \rfloor}, v_i^{\lfloor g/2 \rfloor + 1}) : j \in D_i(y), i \in [n']\}.$$

See Figure 5.2 for a diagram of the construction.

Note that $\text{Girth}(G) = g$ if there exists i such that $D_i(x) \cap D_i(y) \neq \emptyset$, i.e., x and y are not disjoint. However, if x and y are disjoint, then the shortest cycle is of length at least $4 + 2\lfloor \frac{g-2}{2} \rfloor \geq g + 1$. Hence, determining whether the girth is at most g determines whether x and y are disjoint. □

6. Toward fast per-item processing. In section 3, we gave a spanner construction that processes each edge much faster than previous spanner-construction algorithms. In this section, we explore two general methods for decreasing the per-edge computation time of a streaming algorithm. As a consequence, we will show how some results from [20] give rise to efficient graph-stream algorithms.

Our first observation is that we can locally amortize per-edge processing by using some of our storage space as a *buffer* for incoming edges. While the algorithm processes a time-consuming edge, subsequent edges can be buffered subject to the availability of space. This yields a potential decrease in the minimum allowable time between the arrival of consecutive pairs of incoming edges.

THEOREM 6.1. *Consider a streaming algorithm that runs in space $S(n)$ and uses computation time $\tau(m, n)$ to process the entire stream. This streaming algorithm can be simulated by a one-pass streaming algorithm that uses $O(S(n) \log n)$ storage space and has worst-case time per-edge $\tau(m, n)/S(n)$.*

Next we turn to capitalizing on work done to speed up *dynamic graph algorithms.* Dynamic graph algorithms allow edges to be inserted and deleted in any order and the current graph to be queried for a property $\mathcal{P}$ at any point. *Partially dynamic algorithms*, on the other hand, are those that allow only edge insertions and querying. In [20], the authors describe a technique called *sparsification* and use it to speed up existing dynamic graph algorithms that decide whether a graph has property $\mathcal{P}$. Sparsification is based on maintaining strong certificates throughout the updates to the graph.

DEFINITION 6.2. *For any graph property $\mathcal{P}$ and graph G, a* strong certificate *for G is a graph G' on the same vertex set such that, for any H, $G \cup H$ has property $\mathcal{P}$ if and only if $G' \cup H$ has it as well.*

It is easy to see that strong certificates obey a transitivity property: If G' is a strong certificate of property $\mathcal{P}$ for graph G, and G'' is a strong certificate for G', then G'' is a strong certificate for G. Strong certificates also obey a compositional property. If G' and H' are strong certificates of $\mathcal{P}$ for G and H, then $G' \cup H'$ is a strong certificate for $G \cup H$.

In order to achieve their speedup, the authors of [20] ensure that the certificates they maintain are not only strong but also sparse. A property is said to have *sparse certificates* if there is some constant c such that for every graph G on an n-vertex set, we can find a strong certificate for G with at most cn edges. Maintaining $\mathcal{P}$ over a sparse certificate allows an algorithm to run on a dense graph, using the (smaller) computational time required for a sparse graph.

TABLE 6.1
One-pass, $O(n \operatorname{polylog} n)$ space streaming algorithms given by Theorem 6.3.

Problem	Time/edge
Bipartiteness	$\alpha(n)$
Connected comps.	$\alpha(n)$
2-vertex connected comps.	$\alpha(n)$
3-vertex connected comps.	$\alpha(n)$
4-vertex connected comps.	$\log n$
MST	$\log n$
2-edge connected comps.	$\alpha(n)$
3-edge connected comps.	$\alpha(n)$
4-edge connected comps.	$n\alpha(n)$
Constant edge connected comps.	$n \log n$

In the streaming model, we are concerned only with edge insertion and need only query the property at the end of the stream. Moreover, observe that a sparse certificate fits in space $O(n \operatorname{polylog} n)$. The following theorem states that any algorithm that could be sped up via the three major techniques described in [20] yields a one-pass, $O(n \operatorname{polylog} n)$ space, streaming algorithm with the improved running time per input edge.

THEOREM 6.3. *Let $\mathcal{P}$ be a property for which we can find a sparse certificate in time $f(n, m)$. Then there exists a one-pass, semistreaming algorithm that maintains a sparse certificate for $\mathcal{P}$ using $f(n, O(n))/n$ time per edge.*

Proof. Let the edges in the stream be denoted $e_1, e_2, \ldots, e_m$. Let G_i denote the subgraph given by $e_1, e_2, \ldots, e_i$. Inductively, assume we have a sparse certificate C_{jn} for G_{jn}, where $1 \leq j \leq \lfloor m/n \rfloor$, constructed in time $f(n, O(n))/n$ per edge. Also, inductively assume that we have buffered the next n edges, $e_{jn+1}, e_{jn+2}, \ldots, e_{(j+1)n}$. Let $T = C_{jn} \cup \{e_{jn+1}, e_{jn+2}, \ldots, e_{(j+1)n}\}$. By the composability of strong certificates, T is a strong certificate for $G_{(j+1)n}$. Let $C_{(j+1)n}$ be the sparse certificate of T. By the transitivity of strong certificates, $C_{(j+1)n}$ is a sparse certificate of $G_{(j+1)n}$. Since C_{jn} is sparse, $|T| = (c+1)n$. Thus, computing $C_{(j+1)n}$ takes time $f(n, O(n))$. By Theorem 6.1, this results in $f(n, O(n))/n$ time per edge, charged over $e_{jn+1}, e_{jn+2}, \ldots, e_{(j+1)n}$. This computation can be done while the next n edges are being buffered. If $k = n\lfloor m/n \rfloor$, then the final sparse certificate will be $C_k \cup \{e_{k+1}, e_{k+2}, \ldots, e_m\}$. □

We note that, for $f(n, m)$ that is linear or sublinear in m, a better speedup may be achieved by buffering more than n edges, which is possible when we have more space. In [20], the authors provide many algorithms for computing various graph properties which they speed up using sparsification. Applying Theorem 6.3 to these algorithms yields the list of streaming algorithms outlined in Table 6.1. For $l \geq 2$, the l-vertex and l-edge connectivity problems either have not been explicitly considered in the streaming model or have algorithms with significantly slower time per edge [22].

Acknowledgments. We thank Martin Sauerhoff for helpful comments on the results in section 4 and the anonymous referees for their comments on the presentation.

REFERENCES

[1] J. ABELLO, A. L. BUCHSBAUM, AND J. WESTBROOK, *A functional approach to external graph algorithms*, Algorithmica, 32 (2002), pp. 437–458.

[2] G. AGGARWAL, M. DATAR, S. RAJAGOPALAN, AND M. RUHL, *On the streaming model augmented with a sorting primitive*, in Proceedings of the IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 2004, pp. 540–549.

[3] R. Albert, H. Jeong, and A.-L. Barabasi, *The diameter of the world wide web*, Nature, 401 (1999), p. 130.
[4] N. Alon, Y. Matias, and M. Szegedy, *The space complexity of approximating the frequency moments*, J. Comput. System Sci., 58 (1999), pp. 137–147.
[5] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg, *Near-linear time construction of sparse neighborhood covers*, SIAM J. Comput., 28 (1998), pp. 263–277.
[6] Z. Bar-Yossef, R. Kumar, and D. Sivakumar, *Reductions in streaming algorithms, with an application to counting triangles in graphs*, in Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2002, pp. 623–632.
[7] S. Baswana and S. Sen, *A simple linear time algorithm for computing a* $(2k-1)$*-spanner of* $O(n^{1+1/k})$ *size in weighted graphs*, in Proceedings of the International Colloquium on Automata, Languages and Programming, Eindhoven, The Netherlands, 2003, pp. 384–396.
[8] S. Baswana, *Streaming algorithm for graph spanners—single pass and constant processing time per edge*, Inform. Process. Lett., 106 (2007), pp. 110–114.
[9] B. Bollobás, *Random Graphs*, Academic Press, London, 1985.
[10] A. L. Buchsbaum, R. Giancarlo, and J. Westbrook, *On finding common neighborhoods in massive graphs*, Theoret. Comput. Sci., 299 (2003), pp. 707–718.
[11] L. S. Buriol, G. Frahling, S. Leonardi, A. Marchetti-Spaccamela, and C. Sohler, *Counting triangles in data streams*, in Proceedings of the ACM Symposium on Principles of Database Systems, ACM, New York, 2006, pp. 253–262.
[12] A. Chakrabarti, G. Cormode, and A. McGregor, *A near-optimal algorithm for computing the entropy of a stream*, in Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2007, pp. 328–335.
[13] E. Cohen, *Fast algorithms for constructing t-spanners and paths with stretch t*, SIAM J. Comput., 28 (1998), pp. 210–236.
[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press, Cambridge, MA, McGraw-Hill, New York, 2001.
[15] G. Cormode and S. Muthukrishnan, *Space efficient mining of multigraph streams*, in Proceedings of the ACM Symposium on Principles of Database Systems, ACM, New York, 2005, pp. 271–282.
[16] C. Demetrescu, I. Finocchi, and A. Ribichini, *Trading off space for passes in graph streaming problems*, in Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2006, pp. 714–723.
[17] M. Elkin, *Computing almost shortest paths*, ACM Trans. Algorithms, 1 (2005), pp. 283–323.
[18] M. Elkin, *Streaming and fully dynamic centralized algorithms for constructing and maintaining sparse spanners*, in Proceedings of the International Colloquium on Automata, Languages and Programming, Wroclaw, Poland, 2007, pp. 716–727.
[19] M. Elkin and J. Zhang, *Efficient algorithms for constructing* $(1+\epsilon,\beta)$*-spanners in the distributed and streaming models*, Distributed Computing, 18 (2006), pp. 375–385.
[20] D. Eppstein, Z. Galil, G. F. Italiano, and A. Nissenzweig, *Sparsification—a technique for speeding up dynamic graph algorithms*, J. ACM, 44 (1997), pp. 669–696.
[21] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang, *Graph distances in the streaming model: The value of space*, in Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2005, pp. 745–754.
[22] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang, *On graph problems in a semi-streaming model*, Theoret. Comput. Sci., 348 (2005), pp. 207–216.
[23] J. Feigenbaum, S. Kannan, M. J. Strauss, and M. Viswanathan, *An approximate* L^1*-difference algorithm for massive data streams*, SIAM J. Comput., 32 (2002), pp. 131–151.
[24] A. C. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. Strauss, *Fast, small-space algorithms for approximate histogram maintenance*, in Proceedings of the ACM Symposium on Theory of Computing, ACM, New York, 2002, pp. 389–398.
[25] M. Greenwald and S. Khanna, *Efficient online computation of quantile summaries*, in Proceedings of the ACM International Conference on Management of Data, ACM, New York, 2001, pp. 58–66.
[26] S. Guha, N. Koudas, and K. Shim, *Approximation and streaming algorithms for histogram construction problems*, ACM Trans. Database Syst., 31 (2006), pp. 396–438.
[27] M. R. Henzinger, P. Raghavan, and S. Rajagopalan, *Computing on data streams*, in External Memory Algorithms, AMS, Providence, RI, 1999, pp. 107–118.
[28] P. Indyk, *Stable distributions, pseudorandom generators, embeddings and data stream computation*, in Proceedings of the IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 2000, pp. 189–197.

[29] P. Indyk and D. P. Woodruff, *Optimal approximations of the frequency moments of data streams*, in Proceedings of the ACM Symposium on Theory of Computing, ACM, New York, 2005, pp. 202–208.
[30] R. Jain, J. Radhakrishnan, and P. Sen, *A direct sum theorem in communication complexity via message compression*, in Proceedings of the International Colloquium on Automata, Languages and Programming, Eindhoven, The Netherlands, 2003, pp. 300–315.
[31] H. Jowhari and M. Ghodsi, *New streaming algorithms for counting triangles in graphs*, in Proceedings of the International Conference on Computing and Combinatorics, Kunming, Yunnan, China, 2005, pp. 710–716.
[32] B. Kalyanasundaram and G. Schnitger, *The probabilistic communication complexity of set intersection*, SIAM J. Discrete Math., 5 (1992), pp. 545–557.
[33] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins, *Extracting large-scale knowledge bases from the web*, in Proceedings of the International Conference on Very Large Data Bases, Edinburgh, Scotland, 1999, pp. 639–650.
[34] E. Kushilevitz and N. Nisan, *Communication Complexity*, Cambridge University Press, Cambridge, UK, 1997.
[35] F. Lazebnik, V. A. Ustimenko, and A. J. Woldar, *A new series of dense graphs of high girth*, Bull. AMS, 32 (1995), pp. 73–79.
[36] A. McGregor, *Finding graph matchings in data streams*, in Proceedings of APPROX-RANDOM, Berkeley, CA, 2005, pp. 170–181.
[37] S. Muthukrishnan, *Data Streams: Algorithms and Applications*, Now Publishers, Boston, 2005.
[38] N. Nisan and A. Wigderson, *Rounds in communication complexity revisited*, SIAM J. Comput., 22 (1993), pp. 211–219.
[39] A. A. Razborov, *On the distributional complexity of disjointness*, Theoret. Comput. Sci., 106 (1992), pp. 385–390.
[40] M. Thorup and U. Zwick, *Approximate distance oracles*, in Proceedings of the ACM Symposium on Theory of Computing, ACM, New York, 2001, pp. 183–192.
[41] M. Zelke, *k-connectivity in the semi-streaming model*, http://arxiv.org/abs/cs/0608066, 2006.
[42] M. Zelke, *Optimal per-edge processing times in the semi-streaming model*, Inform. Process. Lett., 104 (2007), pp. 106–112.